Lab 10: Graphs and Networks

Exercises 7, 8, 9 and 14, 15 are optional. Exercise 6 is related to the coursework.

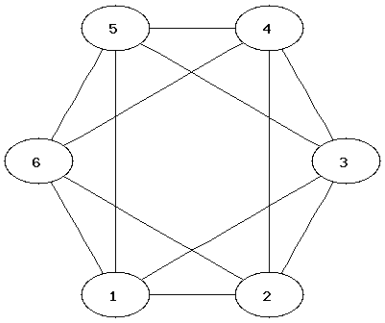
Exercise 14 is possibly the most fun!

Part A: Graph Theory

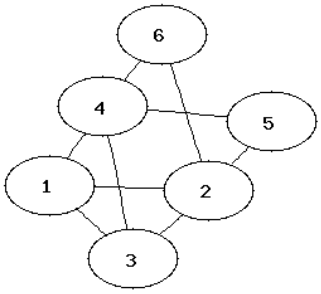
1 Isomorphic graphs

a) Which of the following graphs are isomorphic (i.e. they are the same but their presentation is different)?

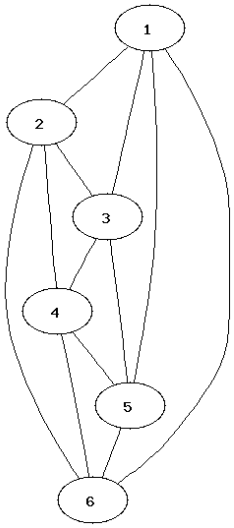
1.



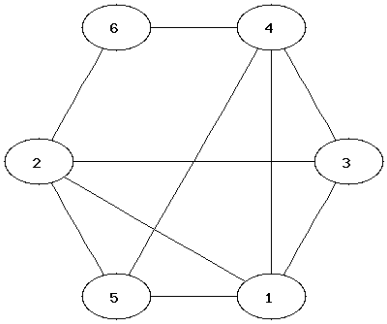
2.



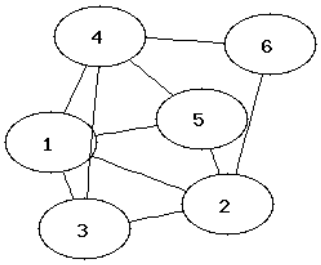
3.



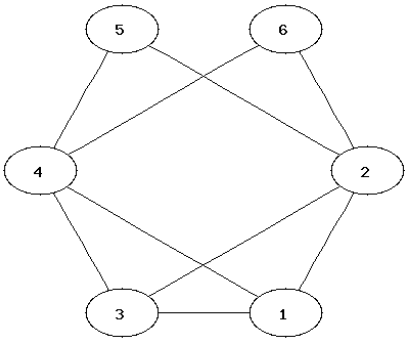
4.



5.



6.



b) It can be easier to determine that two graphs are not isomorphic than to show that they are. Which property of the nodes can be used to determine that two graphs are not isomorphic?

2 Chromatic numbers and Hamiltonian and Eulerian paths

a) The chromatic number of a graph is the minimum number of colours needed to colour a graph so that neighbouring nodes have different colours.
Determine the chromatic number of the graphs from Exercise 1.

b) Which of the graphs in Exercise 1 have an Eulerian path? Which an Eulerian circuit?
Could any node be used for the start or finish node?

c) Which of the graphs in Exercise 1 have a Hamiltonian path?

Hint: If two graphs are isomorphic, what does that imply for their chromatic number and the existence of Hamiltonian or Eulerian paths?

3 Creating Graphs with networkx

Here are the main commands you will need. For further information, refer to the introductory tutorial at https://networkx.github.io/documentation/latest/tutorial/index.html and/or the full documentation for networkx at https://networkx.github.io/documentation/latest/reference/index.html.

To use networkx, it first needs to be imported in Python[1]:

```
import networkx as nx
```

Commands for creating a graph and adding nodes and edges (adding edges will automatically create any nodes that are referred to):

```
G=nx.Graph()                                          # initialise a new graph
G.add_node("One")                                     # add one node
G.add_nodes_from(["Hello", "World"])                  # add a list of nodes
G.add_edge("Hello", "World")                          # add one edge
G.add_edges_from([("One", "Hello"), ("One", "World")])   # add a list of edges
H= DiGraph(G)                                    # create a directed graph H from G
```

Commands for listing nodes and edges:

```
G.nodes()
G.edges()
```

Removing all nodes and edges:

```
G.clear()
```

- Now create graph G as graph 1 from Exercise 1.

4 Visualising graphs with matplotlib

a) Use pyplot of matplotlib to plot the graph G you created in Exercise 3.

Below are the main commands you will need. For further information, refer to the relevant networkx page at https://networkx.github.io/documentation/latest/tutorial/tutorial.html#drawing-graphs (and details at https://networkx.github.io/documentation/latest/reference/drawing.html); matplotlib tutorials at http://www.scipy-lectures.org/intro/matplotlib/matplotlib.html and http://matplotlib.org/users/beginner.html.

To use pyplot, it first needs to be imported in Python[2]:

```
import matplotlib.pyplot as plt
```

The principle is to first DRAW the graph, then SHOW it using pyplot.

```
nx.draw(G)                 # draw graph without labels or axes, default layout
nx.draw_networkx(G)        # draw graph with labels and axes

plt.show()                 # open a graph window to show the plot
plt.savefig("graph.png")   # save the plot as a png file
```

[1] If this gives the error "No module named networkx", you need to install networkx first. In Windows, use pip install.
[2] If this gives the error "No module named matplotlib", you need to install it first. In Windows, use pip install.

By default, matplotlib is not in interactive mode. That means that the graph will not be plotted until you issue the plt.show() command. At this point the python shell will become inactive; you need to close the graph window before you can run another python command. This is useful when you want to draw a complex graph using several draw commands, but it can be annoying.
To run in interactive mode and open one window, issue the commands

```
plt.ion()
plt.figure(1)
```

Any new draw command will add to the existing figure unless you clear the figure first using

```
plt.clf()
```

Alternatively, you can open a second plot window and activate it using

```
plt.figure(2)
```

Using different numbers, you can open lots of windows. Re-issue plt.figure(x) to activate window x again.
Turn off interactive mode with `plt.ioff()`.

b) NetworkX provides many standard graphs. Create these 3 additional graphs:

```
G1=house_x_graph()
G2=complete_bipartite_graph(3,5)
G3=lollipop_graph(10,20)
```

Use the `info(G)` command to print a short summary of the graph properties.

c) Graph Layouts
Networkx provides a choice of five built-in graph layouts:

```
circular_layout, random_layout, spectral_layout, spring_layout, shell_layout.
```

random_layout is the default.
random_layout and spring_layout will generate a different layout every time you use it as they contain random components!
To specify the layout to use, use nx.draw or nx.draw_networkx with the optional “pos” argument.
Examples:

```
nx.draw(G2, pos=spectral_layout(G2))        # draw G2 in spectral layout, no labels
nx.draw_networkx(G3, pos=shell_layout(G3))  # G3 in shell layout, with labels
```

- Try different layouts for the graphs G, G1, G2, G3. (Use the random one several times each!)
- Which layouts are the most suitable for each graph?

5 Other graph operations

NetworkX provides many graph operations. Using the graphs G (from Exercise 3) and G1, G2, G3 from Exercise 4 determine the following:

a) Which of the graphs are bipartite (use `is_bipartite(G)`)?
b) Generate graph G1C as the complement of G1 (use `complement(G1)`). Draw the graphs for G1 and its complement. Can you figure out what is meant by "complement" here?

6 Dependency diagrams/ call graphs for Python applications

"Dependency diagrams" give a visual representation of the structure of a programming project, showing how modules are connected, or how functions are connected by calls (which function calls which other functions). Dependency diagrams are also called "call graphs" or "code maps".

Reasons for creating dependency diagrams:

1. Useful for code testing and debugging
2. Useful for getting to know code written by others (including coursework submissions!)
3. More dependencies (higher complexity) may reduce code efficiency
4. More dependencies can make code harder to maintain
5. More dependencies may also have a good point, showing that code is modular.

Because of 2., you need to submit a dependency diagram for your coursework with the code.

a) In the lecture, you saw two dependency diagrams for different extended implementations of the Caesar Cipher exercise from earlier in the module. Compare the diagrams. You don't have the code itself, but what do these diagrams tell you?
b) Identify any potential issues with the structure of the applications, as shown by the dependency diagrams.
c) There are various methods for creating dependency diagrams or call graphs. Some only look at module level, which is not much use where a module contains several functions. Refer to the instructions given in the separate pdf "Dependency Diagrams" in moodle, and use them to draw a call graph of your own Caesar Cipher implementation.

7 **OPTIONAL** Minimum Spanning Trees with Kruskal's Algorithm

Download the script **kruskal.py** from moodle or copy it from below.

```
# Kruskal's algorithm for finding Minimum Spanning Tree
# for Graphs lecture
# this working script is deliberately not PEP8 compliant
N = 7 # number of nodes
# E contains edges in the format [node, node, weight]
E = [[1, 2, 7], [1, 4, 5], [2, 3, 8], [2, 4, 9],
     [2, 5, 7], [3, 5, 5], [4, 5, 15], [4, 6, 6],
     [5, 6, 8], [5, 7, 9], [6, 6, 11] ]

# Sort the edges, shortest first
E.sort(key=lambda a: a[2])
# Set of nodes that are in the mst
included = set() # ensures nodes can only be added once
# List of edges in MST
mst = []
while len(included)<N:
  h,E = E[0],E[1:] # h is the first edge
  print(f'checking edge {h}, nodes already added: {included}')
  print(f'edges remaining {E}')
  # Ignore edges where both nodes are included
  if h[0] in included and h[1] in included:
    pass # Do nothing - this is already there
```

```
  else:
    # Add both vertices to set of nodes
    included.add(h[0])
    included.add(h[1])
    mst.append(h) # add edge to mst
print (f'MST is {mst}')
```

This is a working script which calculates the MST for the example graph E.

a) Run the script. Inspect the code. Do you understand all of it? A useful resource for this is https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/.

b) Explain the purpose of `E.sort(key=lambda a: a[2])`, and explain how lambda works.

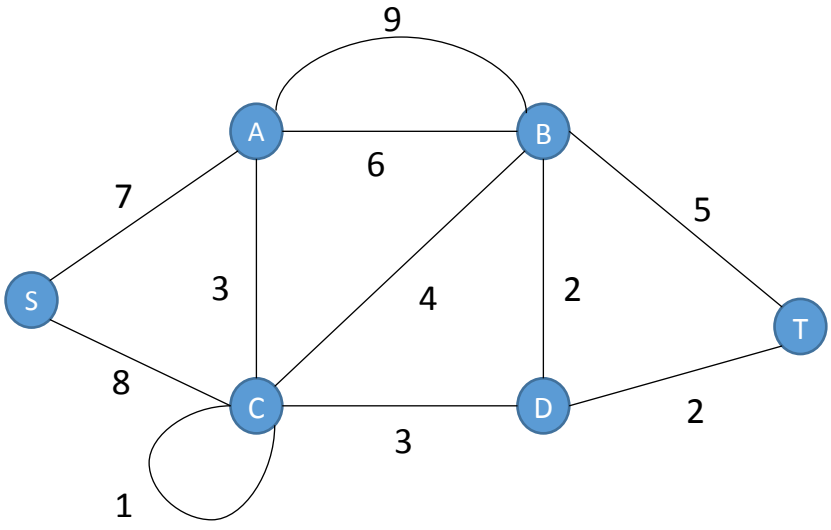
c) The script is not PEP8 compliant. Modify the script to make it PEP8 compliant.
Hint: You can use the 3rd party module "pycodestyle" to check for PEP8 compliance. See https://pycodestyle.readthedocs.io/en/latest/.

d) The script is PEP8 compliant now, but it is still poorly written in many respects.
- N = 7 "hardcodes" the number of edges. This is not "future-proof". Remove this line and instead add code below the definition of the graph E that calculates N from E. Test your code - it should still work e.g. if you remove `[5, 7, 9]` from E, which removes node 7 and therefore changes N.
- Wrap the algorithm itself in a function, move the test case to main() and add a boilerplate.

e) Add code to calculate the "cost" of the MST (i.e. the sum of the length of the edges). Also calculate the "cost" of the original tree and the saving.

f) Add code to draw the graphs of the original graph E and the MST, so that they can be visually compared.

g) Let's call the hand-drawn graph below E1. Run your code with this second test case. Does it work?

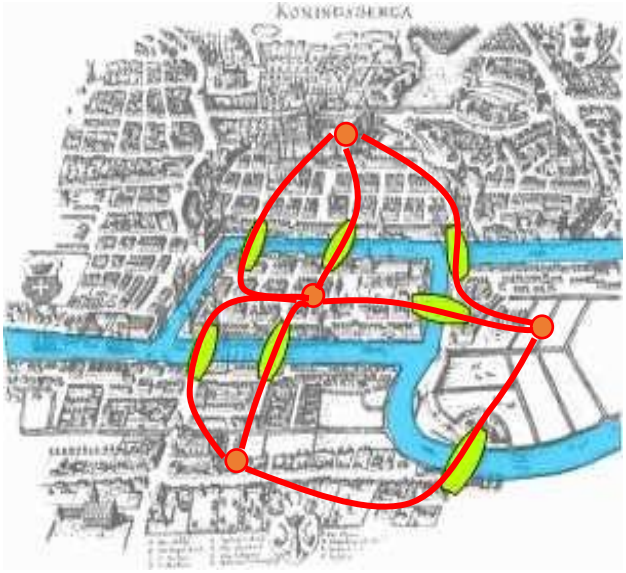


h) Check your answer for part f against the model solution at https://www.tutorialspoint.com/data_structures_algorithms/kruskals_spanning_tree_algorithm.htm.

i) [S, C, 8] is not part of the MST. To what value (assume integers only) would its weight have to be reduced for this edge to be part of the MST? Which Edge would it replace?

8 **OPTIONAL** Back to Königsberg

a) How many bridges have to be added so that the Sunday walk becomes possible? Where should this/these bridge(s) be placed?
b) If one bridge was demolished, would the walk become possible?
c) If yes, could this be any bridge?

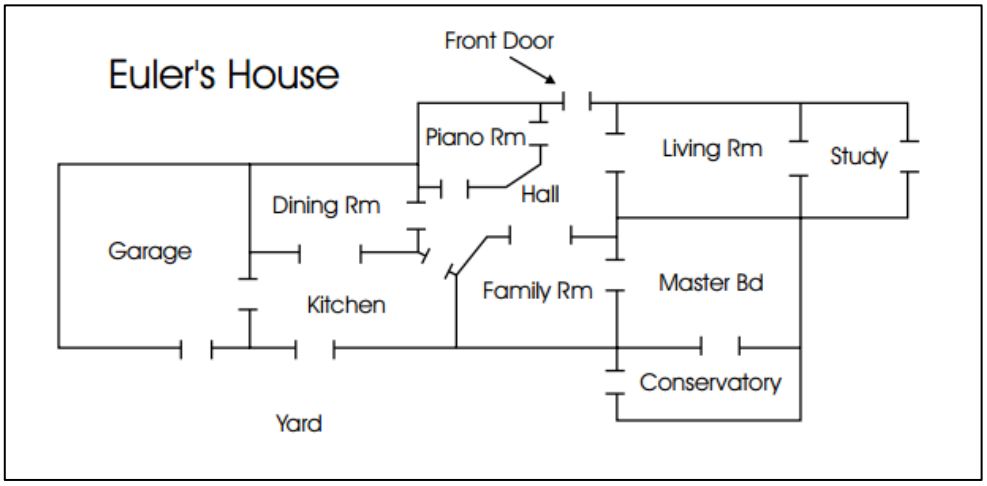


9 **OPTIONAL** Euler's House

(adapted from http://www.math.utah.edu/mathcircle/notes/MC_Graph_Theory.pdf)

Baby Euler has just learned to walk. Use graphs and the floor plan of his house below to answer these questions:

a) He is curious to know if he can walk through every doorway in his house exactly once, and return to the room he started in. Will baby Euler succeed? (note that he is also allowed in the garden / yard)
b) Can baby Euler walk through every door exactly once and return to a different place than where he started?
c) What if the front door is closed?



Part B: Networks

10 The Small-world effect

For this exercise, use graphs G, G1, G2, G3 that you created in NetworkX in Part A.

a) Use `diameter(G)` and `radius(G)` to calculate the diameter and the radius of the graphs.
The diameter is the maximum distance (the longest shortest path) between two nodes in the graph; the radius is the minimum steps needed to reach all nodes from a single node.
b) How much larger can the diameter be compared to the radius?
c) Which of the graphs have a small-world effect with "six degrees of separation" (i.e. a diameter less than 6)?

11 Analysing relations – friends

a) Download the file **friends.csv** from moodle and save it. This is a comma-separated variable (csv) file. You can look at the file if you open it in a text editor. If you double click a csv-file, it will open in a spreadsheet.
b) The file contains a binary relation which can be represented as a directed graph F using:
`F = read_edgelist("friends.csv",delimiter=",",create_using=DiGraph())`
(F is specified as a directed graph because whether someone calls someone else a "friend" may not be symmetric. If you create a csv file yourself, make sure that it does not contain any blank lines at the end of the file.)
c) What is the domain/co-domain for the friends relation?
d) Draw the graph F. Try different layouts. Which layout seems most suitable? Is it easy to see who is considered "friend" by most people and who has the most friends? (note that thick line ends represent arrows)

12 Friends – analysis with PageRank

PageRank is an algorithm that calculates the "relative importance" of nodes in a directed graph. Basically, nodes that have more incoming edges are more important. Edges coming from more important nodes count more than edges coming from less important nodes.

a) Use the code below to calculate the PageRank for each node in F and then print a list of nodes in decreasing order of importance.

```
import operator as op
pg = pagerank(F)
for item in sorted(pg.iteritems(), key=op.itemgetter(1), reverse=True):
     print (item)
```

Who is considered "friend" by most people according to PageRank? Who has the fewest friends?

b) Calculate the reverse graph FR using `FR=F.reverse()`, i.e., if (John, Paul) is in the original, then (Paul, John) is in the reverse graph.

c) Calculate the PageRank of the reverse graph.
Draw both graphs and save them (or take screenshots).
Answer the following questions based on what you observe in the pictures and the PageRank data:

- Why is Claire high in F and low in the reverse graph?
- Why are Helen and Sue highly ranked in F and in the reverse graph?
- What does this mean for websites: how can a website achieve a high PageRank in Google?

13 Small-world networks

A small-world network has a small average shortest path length (eg below 6) [that means it exhibits a small-world effect]. In addition, the average clustering coefficient is larger than the average clustering coefficient of a random graph with the same number of nodes and edges. (The average clustering coefficient determines whether the neighbours of nodes are also connected and thus the graph forms clusters.)

Using:

```
average_shortest_path_length(G)
average_clustering(G)              ### calculate clustering coefficient
gnm_random_graph(n, m)             ### create a random graph with n nodes, m edges
G.to_undirected()                  ### returns an undirected graph from DiGraph
G.number_of_nodes()
G.number_of_edges()
```

a) Determine whether the friends graph is a small-world network.
(hint: you will need to convert it to an undirected graph first)

b) Determine whether the graphs G, G1, G2, G3 are small-world networks (remember to make use of your results from exercise 2).

c) gnm_random_graph(n,m) creates a single random graph every time it's used. How could this affect your answers to parts a and b?

14 **OPTIONAL**: The Six degrees of Kevin Bacon game

More a game than an exercise. Good if you know quite a few actors.

Go to http://oracleofbacon.org/ and play the game.

Can you find any two actors with a Bacon number of more than 3?

- You could try actors from different eras, e.g. Greta Garbo and Rupert Grint
- Or actors from different countries who speak different languages, e.g. Emma Watson (II) and Emil Jannings [Emma Watson (II) is the one who starred in the Harry Potter movies, Emil Jannings was a German silent movie actor]
- What else could you try? Both the above examples still only give a Bacon number of 3! How can you explain these surprisingly close links?

15 **OPTIONAL**: The hand-shake problem

You are hosting a party with your girlfriend/boyfriend. There are three other couples at the party. Several people shake hands.
Obviously nobody shakes hands with themselves or their girlfriend/boyfriend. Nobody shakes hands with the same person more than once. At the end of the party, you ask everybody including your girlfriend/boyfriend how many hands they have shaken. Each person gives a different answer.

Can you figure out how many hands you shook and how many hands your girlfriend/boyfriend shook?

Hints:

- Consider this a graph problem where the nodes are people and the edges are handshakes.

- What is the maximum number of handshakes possible for a single person in this graph? What is the minimum number?
- Because each person gives a different answer, you can figure out what the answers must have been.
- Insert edges into the graph, starting with the person with the most handshakes and then the one with the second most handshakes.